PEER-TO-PEER METADATA MANAGEMENT FOR KNOWLEDGE DISCOVERY APPLICATIONS IN GRIDS*

Gabriel Antoniu[1]
Gabriel.Antoniu@irisa.fr

Antonio Congiusta[2]
acongiusta@deis.unical.it

Sébastien Monnet[1,2]
Sebastien.Monnet@irisa.fr

Domenico Talia[2]
talia@deis.unical.it

Paolo Trunfio[2]
trunfio@deis.unical.it

[1] *IRISA/INRIA*
Campus de Beaulieu, 35042 Rennes cedex, FRANCE
[2] *DEIS, University of Calabria*
Via P. Bucci 41C, 87036 Rende, Italy

Abstract Computational Grids are powerful platforms gathering computational power and storage space from thousands of geographically distributed resources. The applications running on such platforms need to efficiently and reliably access the various and heterogeneous distributed resources they offer. This can be achieved by using metadata information describing all available resources. It is therefore crucial to provide efficient metadata management architectures and frameworks. In this paper we describe the design of a Grid metadata management service. We focus on a particular use case: the Knowledge Grid architecture which provides high-level Grid services for distributed knowledge discovery applications. Taking advantage of an existing Grid data-sharing service, namely JUXMEM, the proposed solution lies at the border between peer-to-peer systems and Web services.

*This work was carried out for the CoreGRID IST project n°004265, funded by the European Commission.

1. Introduction

Computational Grids are powerful platforms gathering computational power and storage space from thousands of resources geographically distributed in several sites. These platforms are large-scale, heterogeneous, geographically distributed and dynamic architectures. Furthermore they contain many types of resources such as software tools, data sources, specific hardware, etc. These resources are spread over the whole platform. Therefore, it is crucial to provide a mean for the applications running on Grids to localize and access the available resources in such large-scale, heterogeneous, dynamic, distributed environment.

Each Grid resource can be described by a metadata item (eg., an XML document). Such a metadata document may contain the 1) the description of a particular resource, 2) its localization and 3) information on the resource usage (eg., command line options of a software tool, format of a data source, protocol used to access a particular node, etc.). Thus, given a resource metadata, it is possible to access the resource. All the metadata items, describing the whole set of resources available in a given Grid have to be managed in an efficient and reliable way especially in large-scale Grids.

In this paper we propose a software architecture of a scalable Grid metadata management service. We focus on a particular use case: metadata management for the Knowledge Grid [7]. The Knowledge Grid is a *service-oriented* software distributed framework that aims to offer high-level Grid services for knowledge discovery applications running on computational Grids. The Knowledge Grid services are built on top of existing, low-level Grid services such as GRAM [12], GridFTP [1] or MDS [11].

Within the Knowledge Grid architecture, metadata provides information about how an object (either a data source or an algorithm) can be accessed. It consists of information on its actual location and on its format (for a data source) or its usage (for an algorithm).

As metadata is actually stored as pieces of data (eg., XML files), they may be treated as such. We take advantage of the good properties exhibited by an already existing Grid data-sharing service, JUXMEM [2, 4], to store and retrieve metadata. We then build a distributed and replicated hierarchical index of available metadata.

In the next section we briefly present the architecture of the Knowledge Grid and focus on its metadata management needs. Section 3 presents the JUXMEM Grid data-sharing service that we use to reliably store and retrieve both resource metadata and the distributed replicated index. Section 4 describes our architecture for a metadata management Grid service tailored for the Knowledge Grid and based on JUXMEM. Finally, Section 5 presents ongoing work and concludes this paper.

2. The Knowledge Grid

2.1 Knowledge discovery in Grids

Nowadays, big companies have to deal with daily generated large amounts of data. They need tools to both store this information and retrieve knowledge from it. Computational Grids [14] offering high computational power and large storage resources can be used to store and process large amounts of data. Furthermore their geographically distributed nature fits well with the companies architecture. Indeed companies data sources and computational power may be spread all over the world.

However, performing knowledge discovery over such a distributed and often heterogeneous architecture, using data sources and data mining algorithms spread over thousands of nodes is not a trivial task. Building and running a distributed knowledge discovery application on a Grid requires high-level services. Data sources to be mined have to be located, furthermore their format has to be discovered somehow (they could be relational databases, text files, etc.). As well, data mining algorithms and software tools have to be localized and their usage has to be known. Then the computations (data mining algorithms running over data sources) have to be scheduled over available Grid nodes. A knowledge discovery application can be complex, consisting in numerous sequential or parallel data mining algorithms working on identical or different data sources. Some data mining algorithm may be run with the data produced by another data mining algorithm, leading to task dependencies, etc.

The Knowledge Grid provides high-level services and a user-friendly interface VEGA [9] that allows a user to easily describe a distributed knowledge discovery application, it then takes care of locating the resources (data sources, algorithms, computational nodes), scheduling tasks, and executing the application. Within the Knowledge Grid, the application designer only has to describe an *abstract execution plan*, with VEGA, he can even do it graphically. An *abstract execution plan* defines at high level the algorithms to be executed and the data sources to be mined. The Knowledge Grid services (called K-Grid services for short thereafter) are responsible to locate the resources and services and instantiate the execution plan which becomes an *instantiated execution plan* like the one presented in Figure 1.

An instantiated execution plan contains a set of tasks -with assigned Grid resources- to be done (data transfers and computations). It is executed by the K-Grid services and it may be refined as resources may become available or unavailable in a Grid.

```
<ExecutionPlan type="instantiated">
 <Task label="task1">
  <Program href="minos.cs.icar.cnr.it/software/DB2Extractor.xml"
           title="DB2Extractor on minos.cs.icar.cnr.it"/>
  <Input href="minos.cs.icar.cnr.it/data/car-imports_db2.xml"
         title="car-imports.db2 on minos.cs.icar.cnr.it"/>
  <Output href="minos.cs.icar.cnr.it/data/imports-85c_db2.xml"
          title="imports-85c.db2 on minos.cs.icar.cnr.it"/>
 </Task>
 <Task label="check1">
  <ResourceCheck method="soft"/>
 </Task>
 <Task label="task2">
  <Program href="minos.cs.icar.cnr.it/software/GridFTP.xml"
           title="GridFTP on minos.cs.icar.cnr.it"/>
  <Input href="minos.cs.icar.cnr.it/data/imports-85c_db2.xml"
         title="imports-85c.db2 on minos.cs.icar.cnr.it"/>
  <Output href="abstract_host1/data/imports-85c_db2.xml"
          title="imports-85c.db2 on abstract_host1"/>
 </Task>
 ...
 <Task label="task6">
  <Program href="abstract_host1/software/autoclass3-3-3.xml"
           title="autoclass on abstract_host1"/>
  <Input href="abstract_host1/data/imports-85c_db2.xml"
         title="imports-85c.db2 on abstract_host1"/>
  <Output href="abstract_host1/data/classes.xml"
          title="classes on abstract_host1"/>
 </Task>
 ...
 <TaskLink ep:from="task1" ep:to="check1"/>
 <TaskLink ep:from="check1" ep:to="task2"/>
 <TaskLink ep:from="task2" ep:to="task3"/>
 ...
 <TaskLink ep:from="task5" ep:to="task6"/>
 ...
 <ResourceInstantiation abstractResource="abstract_host1">
   <candidateResource>icarus.cs.icar.cnr.it</candidateResource>
   <candidateResource>telesio.cs.icar.cnr.it</candidateResource>
 </ResourceInstantiation>
</ExecutionPlan>
```

Figure 1. A sample instantiated execution plan (from [15]).

2.2 The Knowledge Grid architecture

The K-Grid services are organized in a two-layer software architecture: 1) the High-level K-Grid layer and 2) the Core-level K-Grid layer. In its current implementation, the different services composing the Knowledge Grid are Grid services interacting by using the WSRF [10] standard. The organization of the K-Grid services is described by Figure 2. The High-level K-Grid layer includes services to compose, validate and execute distributed knowledge discovery applications. The main services of the High-level K-Grid services are:

- The **Data Access Service (DAS)**, responsible for data sources and mining results publication and search.

- The **Tools and Algorithms Access Service (TAAS)**, responsible for data mining and visualization tools and algorithms publication and search.

- The **Execution Plan Management Service (EPMS)**, allowing to describe a distributed knowledge discovery application by building an execution graph with constraints on resources. It generates an abstract execution plan (resources are not know yet).

- The **Results Presentation Service (RPS)**, offering services for knowledge discovery results presentation;

The services exhibited by the Core K-Grid layer are:

- The **Knowledge Discovery Service (KDS)**, responsible for metadata management. Every resource (nodes, algorithms and tools, data sources and mining results) of the Knowledge Grid is described by a metadata item. In the Knowledge Grid, resource metadata is a XML document stored in a *Knowledge Metadata Repository* (KMR).
- The **Resource Allocation and Execution Management Service (RAEMS)**, responsible to instantiate an abstract execution plan. It uses the KDS service to find resources satisfying the constraints imposed by the abstract execution plan. It is also responsible for the application execution management.

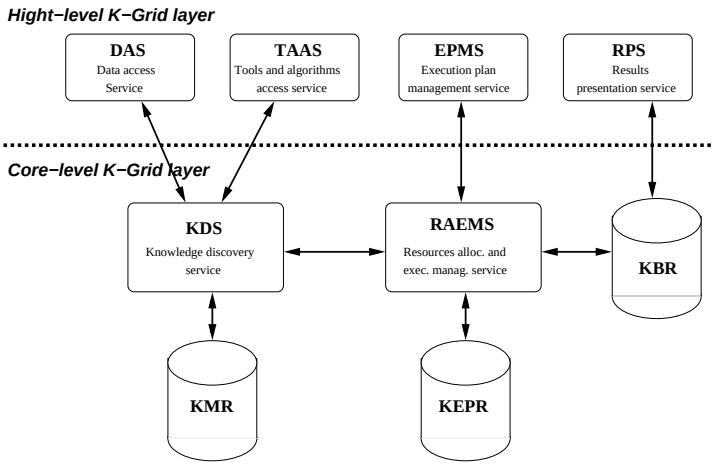

Figure 2. The Knowledge Grid software architecture.

2.3 Current KDS design and limitations

The Knowledge Directory Service (KDS) is responsible for handling metadata describing Knowledge Grid resources. A sample metadata is presented by Figure 3. Such resources include hosts, data repositories, tools and algorithms used to extract, analyze, and manipulate data, execution plans, and knowledge models obtained as result of mining processes.

The metadata information is represented by XML documents stored in a component called Knowledge Metadata Repository (KMR). The functionalities

```
<DataMiningSoftware name="AutoClass">
  <Description>
    <KindOfData>flat file</KindOfData>
    <KindOfKnowledge>clusters</KindOfKnowledge>
    <KindOfTecnique>statistics</KindOfTecnique>
    <DrivingMethod>autonomous knowledge miner</DrivingMethod>
  </Description>
  <Usage>
    ...
    <Syntax>
      <Arg description="executable" type="required" value="/usr/autoclass/autoclass">
        <Arg description="make a classification" type="alternative" value="-search">
          <Arg description="a .db2 file" type="required"/>
          <Arg description="a .hd2 file" type="required"/>
          <Arg description="a .model file" type="required"/>
          <Arg description="a .s-params file" type="required"/>
        </Arg>
        <Arg description="create a report" type="alternative" value="-reports">
          <Arg description="a .results-bin file" type="required"/>
          ...
        </Arg>
        ...
      </Arg>
    </Syntax>
    <Hostname>icarus.cs.icar.cnr.it</Hostname>
    <ManualPath>/usr/autoclass/read-me.text</ManualPath>
    <DocumentationURL>http://ic-www.arc.nasa.gov/ic/projects/...</DocumentationURL>
    ...
  </Usage>
</DataMiningSoftware>
```

Figure 3. An extract from an XML metadata sample for the AutoClass software (presented in [15]).

of the KDS are mostly used by DAS and TAAS services while publishing and searching for datasets and tools to be used in a KDD application. DAS and TAAS services always interact with a local instance of the KDS, which in turn may invoke one or more other remote KDS instances.

The KDS exports three main operations:

- `publishResource`, used to publish metadata related to a given resource into the KMR;

- `searchResource`, for locating resources that match some given search criteria;

- `retrieveMetadata`, invoked to retrieve metadata associated to a given resource identified by a provided KDS URL.

It should be noted that when a `publishResource` is performed, only an interaction between a DAS/TAAS service and the local KDS is needed, because each KMR instance stores metadata about resources available on the same Grid node on which the KMR itself is hosted. On the contrary, when a `searchResource` is invoked, the related query is first dispatched from the DAS/TAAS to the co-located KDS service, which then answers by checking the local KMR, and in turn forwards the same query to remote KDSs with the aim of finding more matches.

The `retrieveMetadata` receives a KDS URL returned by a previous invocation of the `searchResource` operation, and uses it to contact the remote KDS on which the resource is available to retrieve the associated metadata document.

It appears clear, thus, that the `searchResource` is the most complex activity performed by the KDS, because it involves interactions and coordination with remote instances of the same service. On the other hand, it should be mentioned that the architecture of the Knowledge Grid does not prescribe any particular mode of interaction and/or protocol between the different KDS instances.

The current implementation, for instance, is adopting to such purpose one of the simplest strategies: the query forwarding is performed by contacting concurrently all of the known remote KDS instances (avoiding loops).

In this paper we propose a new KDS design based on a shared distributed index handled by a Grid data-sharing service and a peer-to-peer technique. This is useful for reducing the number of remote KDS instances contacted when forwarding a search query.

Resources metadata like the one presented by Figure 3 should be stored in a persistent and fault tolerant storage. Furthermore, they may be shared by multiple applications, and sometimes updated. Therefore, it is necessary to maintain the consistency between the different copies that may exist in the Grid. Thus we use a data-sharing service, JUXMEM, which offers transparent access to persistent mutable data, to store the XML files corresponding to pieces of metadata.

3. JUXMEM: a Grid data-sharing service

In this section we present the JUXMEM Grid data-sharing service used in the design of the metadata management Grid service.

3.1 A hierarchical architecture

From the metadata management Grid service perspective, JUXMEM is a service providing transparent access to persistent, mutable, shared data. When allocating memory, a client has to specify in how many sites[1] the data should be replicated, and on how many nodes in each site. This results into the instantiation of a set of data replicas, associated to a group of peers called *data group*. Usually each node runs one single peer. The allocation primitive returns a global *data-ID*, which can be used by the other nodes to identify existing data. To obtain read and/or write access to a data block, the clients only need to use this data-ID.

The data group is hierarchically organized, as illustrated on Figure 4: the *Global Data Group (GDG)* gathers all provider nodes holding a replica of the same piece of data. These nodes can be distributed in different sites, thereby

[1] A site is a set of clustered nodes, it can be a physical cluster within a cluster federation, or close from a latency viewpoint.

increasing the data availability if faults occur. The GDG is divided into *Local Data Groups (LDG)*, which correspond to data copies located in a same site.

In order to access a piece of data, a client has to be attached to a specific LDG (to "map" the data). Then, when the client performs the read/write and synchronization operations, the consistency protocol layer manages data synchronization and data transmission between clients, LDGs and GDG, within the strict respect of the consistency model.

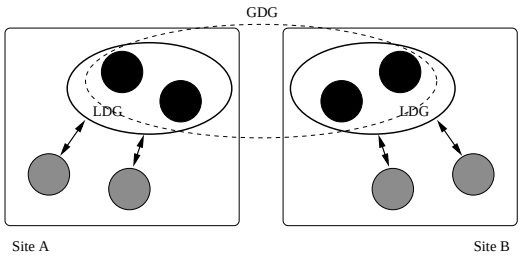

Figure 4. JUXMEM: a hierarchical architecture.

3.2 JUXMEM software architecture

The JUXMEM Grid service is composed of a set of layers presented in Figure 5. The lower layer Juk is the JUXMEM kernel. It relies on JXTA [16] to offer to the uppers layers publish/subscribe operations, efficient communication and storage facilities. Every node involved or using JUXMEM is therefore managed in a peer-to-peer way using JXTA. JXTA is is a set of protocols allowing nodes (Grid nodes, PDA, etc.) to communicate and collaborate in a P2P manner. The implementations of these protocols provide the ability to obtain efficient communications on Grids [6].

Above Juk, a fault-tolerance layer is responsible for hierarchical data replication. It offers the concept of *Self-Organizing Group* (SOG), a SOG is a replication group that is able to adapt itself in case of dynamic changes (by creating new replicas or removing old ones), this provides the ability to keep fault tolerance guarantees even in presence of failures.

The upper layer is responsible for data consistency management, it serves data access requests, manages locks and maintain pending requests lists.

A multi-protocol architecture. The layers presented above are built as interchangeable software modules. Therefore, it is possible for each data item stored by the Grid data-sharing service to specify a particular consistency protocol or a particular SOG implementation.

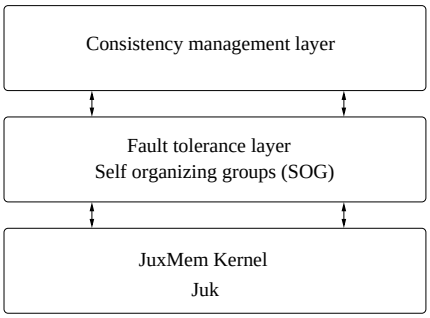

Figure 5. JUXMEM layered software architecture.

The JUXMEM service can not be used in its current design to manage metadata, but additional features must be provided. Data stored in JUXMEM is accessible (localizable) *only* by using its associated data-ID. Metadata items have to be localizable using only names and attributes. The following section presents our approach to build a metadata management service using the JUXMEM data-sharing service.

4. A Grid metadata management service

From the Knowledge Grid viewpoint, the Grid metadata management service consists of a particular design and implementation for the KDS service and the KMR repository. The service presented below serves requests from the TAAS and the DAS High-level K-Grid services but also from the RAEMS Core K-Grid service (see Figure 2).

Our approach relies on the use of the JUXMEM Grid data-sharing service prototype presented in the previous section. Metadata items are stored within the Grid data-sharing service.

4.1 Metadata storage and retrieval

Requirements. Resources metadata should remain available in the Grid. Therefore they should be stored in a *fault tolerant* and *persistent* manner. This may provide the ability to access metadata information in spite of failures and disconnections. Furthermore, some metadata should be *updatable*. In the Knowledge Grid use case, a piece of metadata may describe the result of a knowledge discovery task, this result may be refined later which leads to metadata modifications. If a resource location is changing, it should also be reflected by updating the associated metadata. Finally, metadata has to be localizable by providing name, attributes and constraints upon the described resource.

Storing metadata in a Grid data-sharing service. To achieve high availability of metadata despite failures we store them in the JUXMEM Grid data-sharing service. Each metadata item describing a resource is though replicated and associated to one unique ID as described in Section 3. This ID can then be used to retrieve metadata information stored in the Grid data-sharing service. Availability and consistency (eg., in case of concurrent updates) is then also managed by the Grid data-sharing service. The metadata items that will not be updated (e.g. describing a large data source that will not be updated and will not be moved) can take advantage of JUXMEM multi-protocol feature by using a very simple and efficient consistency protocol without synchronization operations. Thus, JUXMEM is used as a *fault-tolerant, distributed* and *shared* KMR (see Section 2) and JUXMEM's data-IDs are used as KDS URLs.

Locality. Metadata information is strongly linked with the resource it describes. Therefore if the resource becomes unavailable, its corresponding metadata information would become useless (it can also become misleading). Therefore, regarding JUXMEM hierarchical architecture, metadata information should be stored within the site containing the resource it describes (i.e. over one unique JUXMEM LDG). If all the nodes of the site fail (due to a power failure in a computer room for instance) the resources metadata of this site become unavailable but it is also the case of the described resources. Thus, we choose to store metadata information in the described resources' site using only one JUXMEM LDG per metadata item. However notice that LDG are reliable self-organizing groups, ie. the failure of a node does not lead to the loss of metadata items.

4.2 Fault-tolerant distributed indexes

While looking for metadata information using the `searchResource` operation, applications[2] can provide information like a name (eg., a data source name "clientdata1" or an algorithm name "J48") or a set of attributes and constraints as the one in the "Description" section of the metadata presented in Figure 3. An accurate description of the kind of requests the KDS service should be able to serve is given in [15].

Therefore it is necessary to have a mean to find a metadata identifier (which then permit to retrieve the metadata information itself) using names and attributes that represent the resource described by the metadata.

Distributed indexes. Usual approaches rely on the use of a centralized indexing system. It can be either a relational database like MySQL [18] or a

[2]In our current use case the applications are the DAS, TAAS and RAEMS K-Grid services.

LDAP [13] server (used in previous Knowledge Grid implementations). We use distributed indexes: in each site composing the Grid, we maintain a *site index* of the published resources metadata within this site. This site index contains tuples consisting of the resource name, attributes (as a byte vector) and the resource metadata identifier (its JUXMEM data-ID).

Fault tolerance. There again we rely on the JUXMEM data-sharing service: the site indexes are data item that can be stored in JUXMEM. Therefore they are automatically replicated for fault tolerance. Notice that a site index only contains information of its own site, furthermore it does not contain the whole metadata information but only metadata item names and some relevant attributes. Thus, a site index size remains limited.

Index sharing. The WSRF KDS instances serving `publishResource` and `searchResource` requests are clients of the JUXMEM service. In each site it is possible to have multiple KDS services having mapped the site's index as illustrated in Figure 6. The KDS are responsible for parsing the index, finding the metadata identifier, fetching the metadata (using the identifier) and sending back the retrieved metadata to the requester (either DAS, TAAS or RAEMS). These tasks are achieved by interacting with the JUXMEM service. It is important to notice that the site index is a data item stored by JUXMEM and mapped by the multiple KDS: the consistency of the shared index is ensured by the grid data-sharing service while new publications occur.

The shared site indexes allows KDS instances to retrieve *locally* (on their node) KDS URLs of metadata describing resources spread over their site nodes.

4.3 Big picture

4.3.1 Metadata publication. Metadata publications done by the DAS and the TAAS are made through the `publishResource` operation provided by KDS. When a KDS receives such a request:

1 It stores the corresponding XML file within JUXMEM,

2 locks the index to ensure no concurrent publish occurs,

3 updates it, adding the new resource metadata index entry (attributes and JUXMEM data-ID of the XML file).

The KDS then releases the lock upon the index. To allow the other KDS to continue serving search requests while an update occurs, we use a particular consistency protocol allowing read operations concurrent to a write operation. Such a protocol is available in JUXMEM, it is described and evaluated in details in [3]. Note that the publication of a site resource affects only the index stored

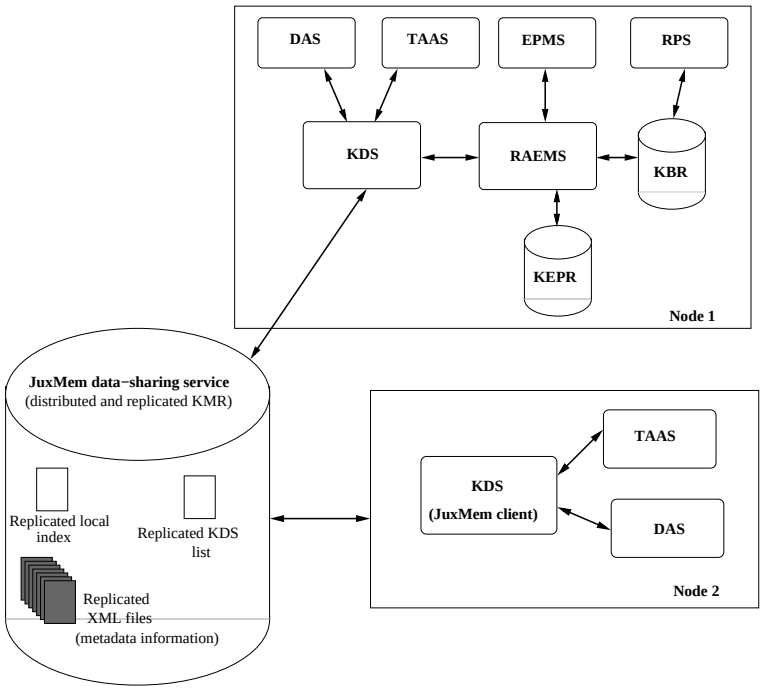

Figure 6. In each site KDS interact with the JUXMEM service to access 1) the local index, 2) the KDS list, and 3) the metadata information itself.

in this site and used by the local KDSs, furthermore resources metadata are also stored on intra-site JUXMEM providers (in LDGs). Therefore a publish operation does not imply inter-site communications.

4.3.2 Metadata search. When K-Grid services need to search a particular resource metadata, they request the KDS running on the same node or a randomly chosen remote KDS within their own site[3]. The KDS receiving such a request search in its mapped (in its local memory) site index. If the resource is found, it gets the corresponding XML file using the data-ID stored in the site index. In this case the resource is available within the same site. If a corresponding resource can not be found in the site index, the KDS forwards it to one randomly chosen KDS in each other site involved in the Grid using JXTA peer-to-peer communication layers as illustrated by Figure 7. To achieve this, a partial list of KDS instances is stored and maintained within the JUXMEM Grid

[3]A round robin policy could also be used.

data-sharing service. This list is replicated hierarchically in the whole platform using the GDG/LDG hierarchy presented in Section 3.

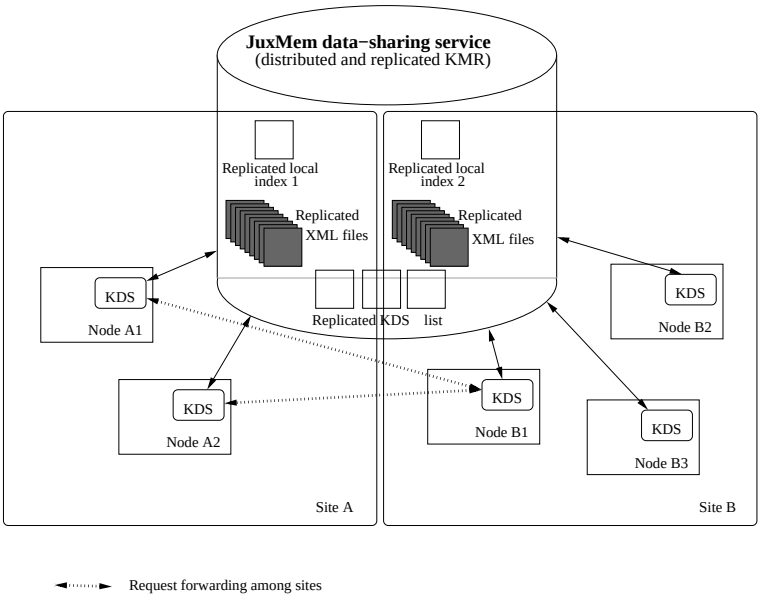

Figure 7. Among sites, KDS cooperate using the hierarchically replicated KDS list.

At initialization, a KDS maps its site index and the KDS list, it can then add itself into this list. If a KDS does not answer to a request (either publish or search), it is removed from this list. The KDS list is not expected to be frequently updated as Grid nodes are assumed more stable than peers in peer-to-peer systems.

4.4 Technical concerns

From a technical view point our solution implies integrating the JUXMEM and the Knowledge Grid research prototypes. JUXMEM entities are managed in a peer-to-peer manner, using Sun Microsystems JXTA protocols, while the entities involved in the Knowledge Grid service use the WSRF Grid standard.

The junction between the two different sets of protocols is done by the new KDS implementation: KDS instances are part of both the JUXMEM platform, as clients of the JUXMEM Grid service, and they serve WSRF requests from the Knowledge Grid services (`publishResource` and `searchResource`).

The KDSs are also responsible to parse the distributed index. The distributed nature of the index implies a cooperation between the KDS instances distributed in different sites all over the Grid. This cooperation is made is a peer-2-peer

manner, taking advantage of the Grid data-sharing service to store, manage and share a neighbor list (the KDS list).

5. Conclusion

Metadata data management in large scale, heterogeneous, geographically distributed and dynamic architectures such as computational Grids is an important problem. Providing an efficient and reliable metadata management service allows applications to easily access heterogeneous resources spread over thousands of nodes.

The solution we presented in this paper takes advantage of already existing work in Grids. By integrating the JUXMEM Grid data-sharing service in the design of a metadata management service for the Knowledge Grid, XML metadata files are stored on a fault tolerant and consistent support, and are kept close to the resources they describe. The proposed two-level index hierarchy allows the applications to get resources located in their own site if they exist or in remote ones otherwise, enhancing locality.

The integration of the two research prototypes is in progress, and we plan to evaluate this solution on a real Grid platform such as Grid'5000 [17, 8]. The format of the distributed index should then be further investigated, using binaries trees for instance. The peer-to-peer cooperation between KDS instances should also be enhanced, for instance by selecting several KDSs for inter-site cooperation.

References

[1] Bill Allcock, Joe Bester, John Bresnahan, Ann L. Chervenak, Carl Kesselman, Sam Meder, Veronika Nefedova, Darcy Quesnel, Steven Tuecke, and Ian Foster. Secure, Efficient Data Transport and Replica Management for High-Performance Data-Intensive Computing. In *Proceedings of the 18th IEEE Symposium on Mass Storage Systems (MSS 2001), Large Scale Storage in the Web*, page 13, Washington, DC, USA, 2001. IEEE Computer Society.

[2] Gabriel Antoniu, Marin Bertier, Eddy Caron, Frédéric Desprez, Luc Bougé, Mathieu Jan, Sébastien Monnet, and Pierre Sens. *Future Generation Grids*, chapter GDS: An Architecture Proposal for a Grid Data-Sharing Service, pages 133–152. CoreGRID series. Springer-Verlag, 2006.

[3] Gabriel Antoniu, Loïc Cudennec, and Sébastien Monnet. Extending the entry consistency model to enable efficient visualization for code-coupling grid applications. In *6th IEEE/ACM International Symposium on Cluster Computing and the Grid*, pages 552–555, Singapore, May 2006. CCGrid 2006.

[4] Gabriel Antoniu, Jean-François Deverge, and Sébastien Monnet. How to bring together fault tolerance and data consistency to enable grid data sharing. *Concurrency and Computation: Practice and Experience*, (17), 2006. To appear.

[5] Gabriel Antoniu, Philip Hatcher, Mathieu Jan, and David A. Noblet. Performance Evaluation of JXTA Communication Layers. In *Proceedings of the Workshop on Global and Peer-to-Peer Computing (GP2PC 2005)*, Cardiff, UK, May 2005. Held in conjunction

with the 5th IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGrid '05). Best presentation award.

[6] Gabriel Antoniu, Mathieu Jan, and David A. Noblet. Enabling the P2P JXTA Platform for High-Performance Networking Grid Infrastructures. In *Proceedings of the 1st International Conference on High Performance Computing and Communications (HPCC '05)*, number 3726 in Lecture Notes in Computer Science, pages 429–440, Sorrento, Italy, September 2005. Springer.

[7] Mario Cannataro and Domenico Talia. The knowledge grid. *Commun. ACM*, 46(1):89–93, 2003.

[8] Franck Cappello, Eddy Caron, Michel Dayde, Frédéric Desprez, Emmanuel Jeannot, Yvon Jegou, Stéphane Lanteri, Julien Leduc, Nouredine Melab, Guillaume Mornet, Raymond Namyst, Pascale Primet, and Olivier Richard. Grid'5000: A Large Scale, Reconfigurable, Controlable and Monitorable Grid Platform. In *Proceedings of the 6th IEEE/ACM International Workshop on Grid Computing (Grid '05)*, Seattle, Washington, November 2005.

[9] Antonio Congiusta, Domenico Talia, and Paolo Trunfio. VEGA: A Visual Environment for Developing Complex Grid Applications. In *Proc. of the First International Workshop on Knowledge Grid and Grid Intelligence (KGGI 2003)*, pages 56–66, Halifax, Canada, October 2003. Department of Mathematics and Computing Science, Saint Mary's University. ISBN 0-9734039-0-X.

[10] Antonio Congiusta, Domenico Talia, and Paolo Trunfio. Distributed data mining services leveraging WSRF. *Future Generation Computer Systems*, 23(1):34–41, January 2007.

[11] Karl Czajkowski, Steven Fitzgerald, Ian Foster, and Carl Kesselman. Grid Information Services for Distributed Resource Sharing. In *Proceedings of the Tenth IEEE International Symposium on High-Performance Distributed Computing*, pages 181–184, San Francisco, CA, August 2001. IEEE Press.

[12] Karl Czajkowski, Ian Foster, Nick Karonis, Cral Kesselman, Stewart Martin, Warren Smith, and Steve Tuecke. Resource Management Architecture for Metacomputing Systems. In *Proc. IPPS/SPDP:Workshop on Job Scheduling Strategies for Parallel Processing*, pages 62–82, March 1998.

[13] Jeff Hodges and Robert Morgan. Lightweight Directory Access Protocol (v3): Technical Specification. IETF Request For Comment 3377, Network Working Group, 2002.

[14] Carl Kesselman and Ian Foster. *The Grid: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann Publishers, November 1998.

[15] Carlo Mastroianni, Domenico Talia, and Paolo Trunfio. Metadata for managing grid resources in data mining applications. *Journal of Grid Computing*, 2(1):85–102, March 2004.

[16] Bernard Traversat, Ahkil Arora, Mohamed Abdelaziz, Mike Duigou, Carl Haywood, Jean-Christophe Hugly, Eric Pouyoul, and Bill Yeager. Project JXTA 2.0 Super-Peer Virtual Network. `http://www.jxta.org/project/www/docs/JXTA2.0protocols1.pdf`, May 2003.

[17] Grid'5000 Project. `http://www.grid5000.org/`.

[18] MySQL. `http://www.mysql.com/`.